

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

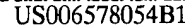
Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problems Mailbox.**



(10) **Patent No.:** US 6,578,054 B1
(45) **Date of Patent:** *Jun. 10, 2003

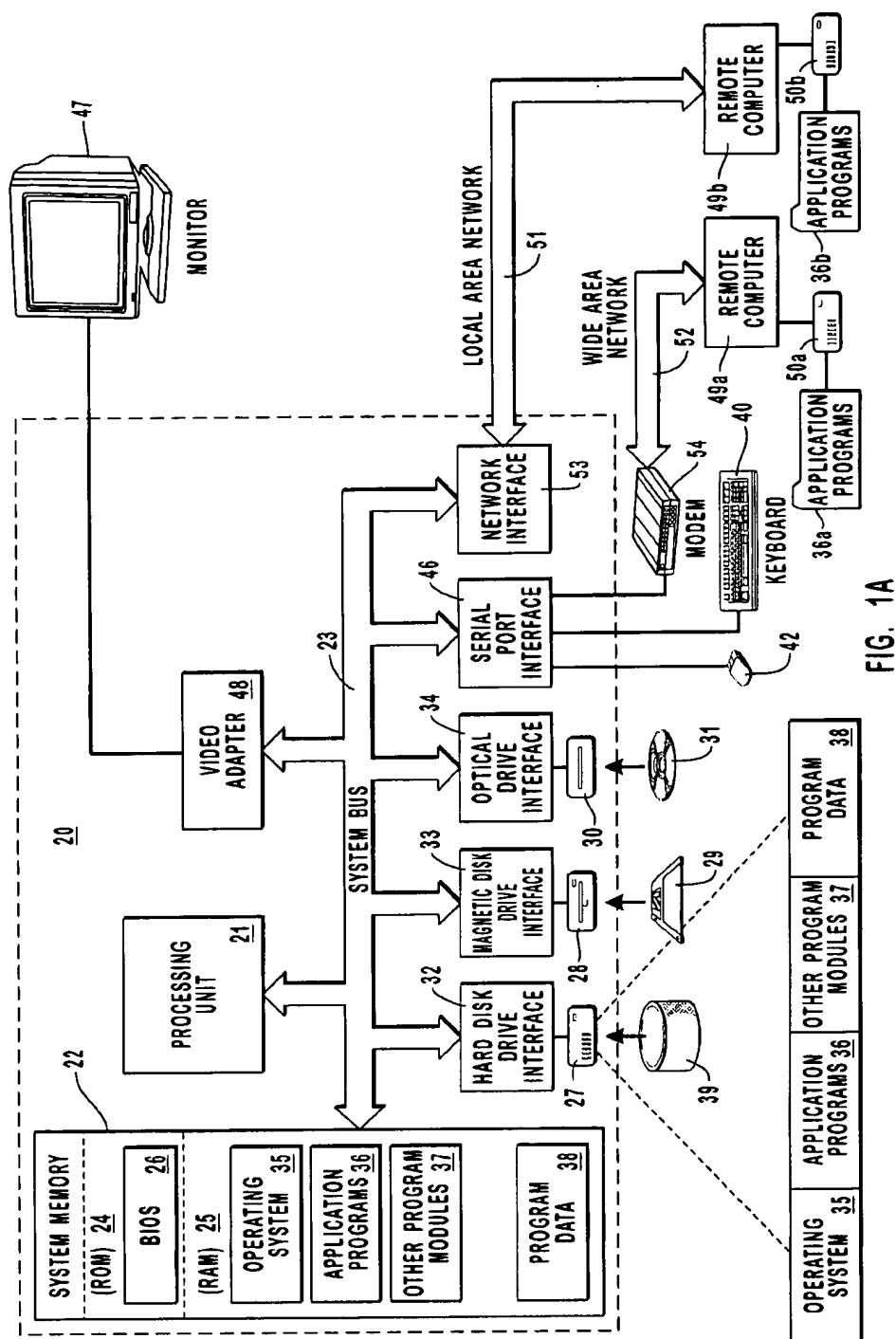


FIG. 1A

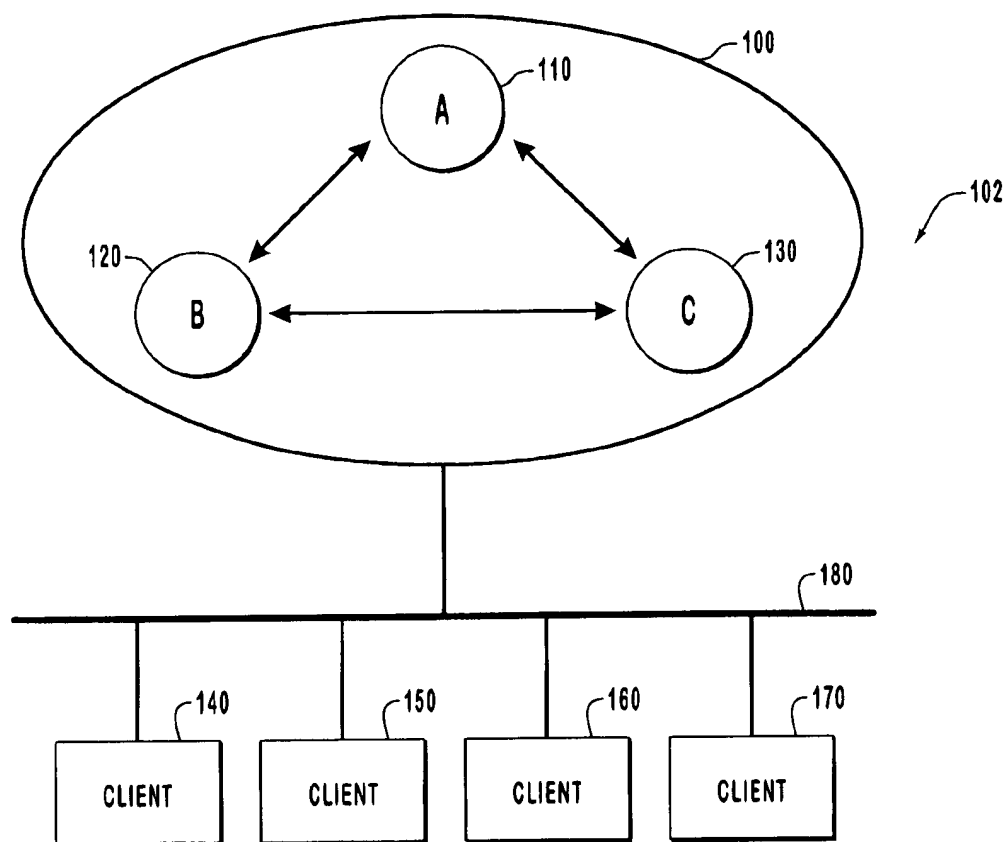


FIG. 1B

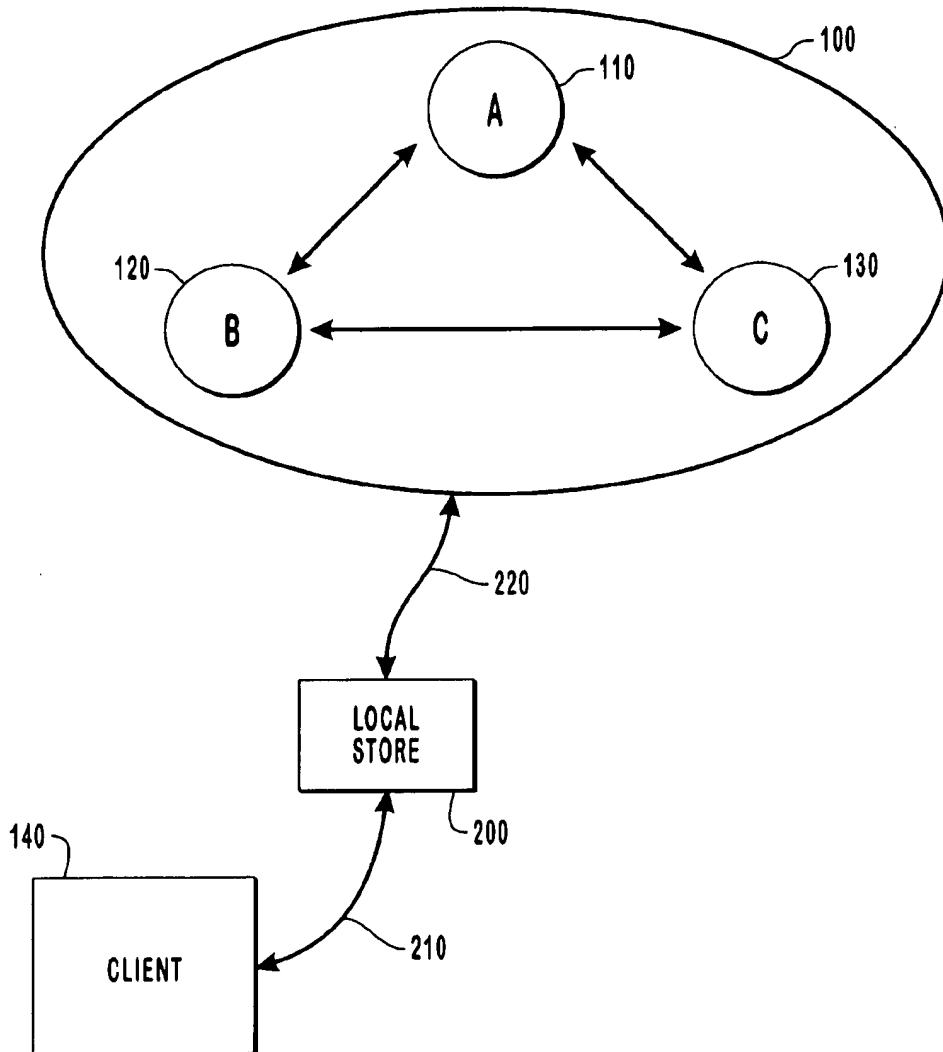


FIG. 2

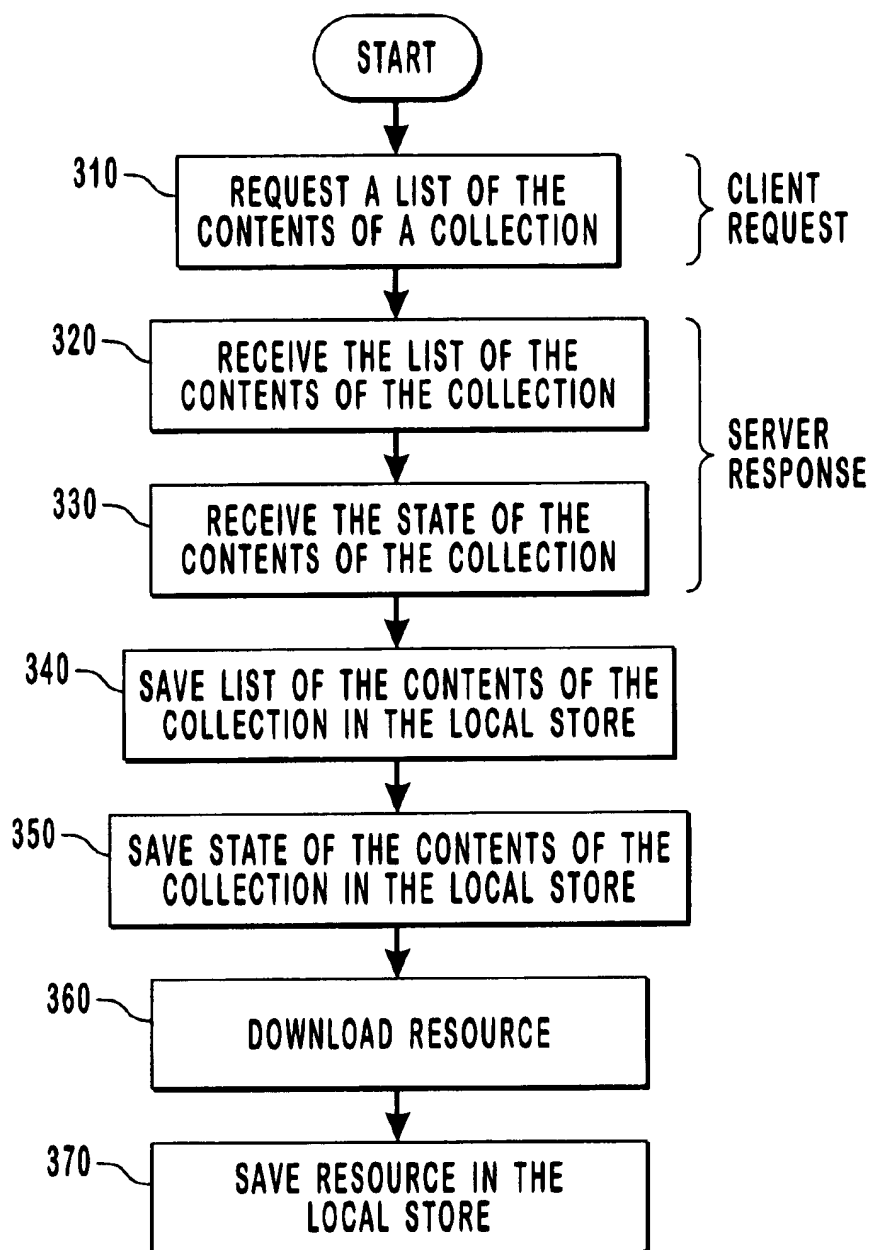


FIG. 3

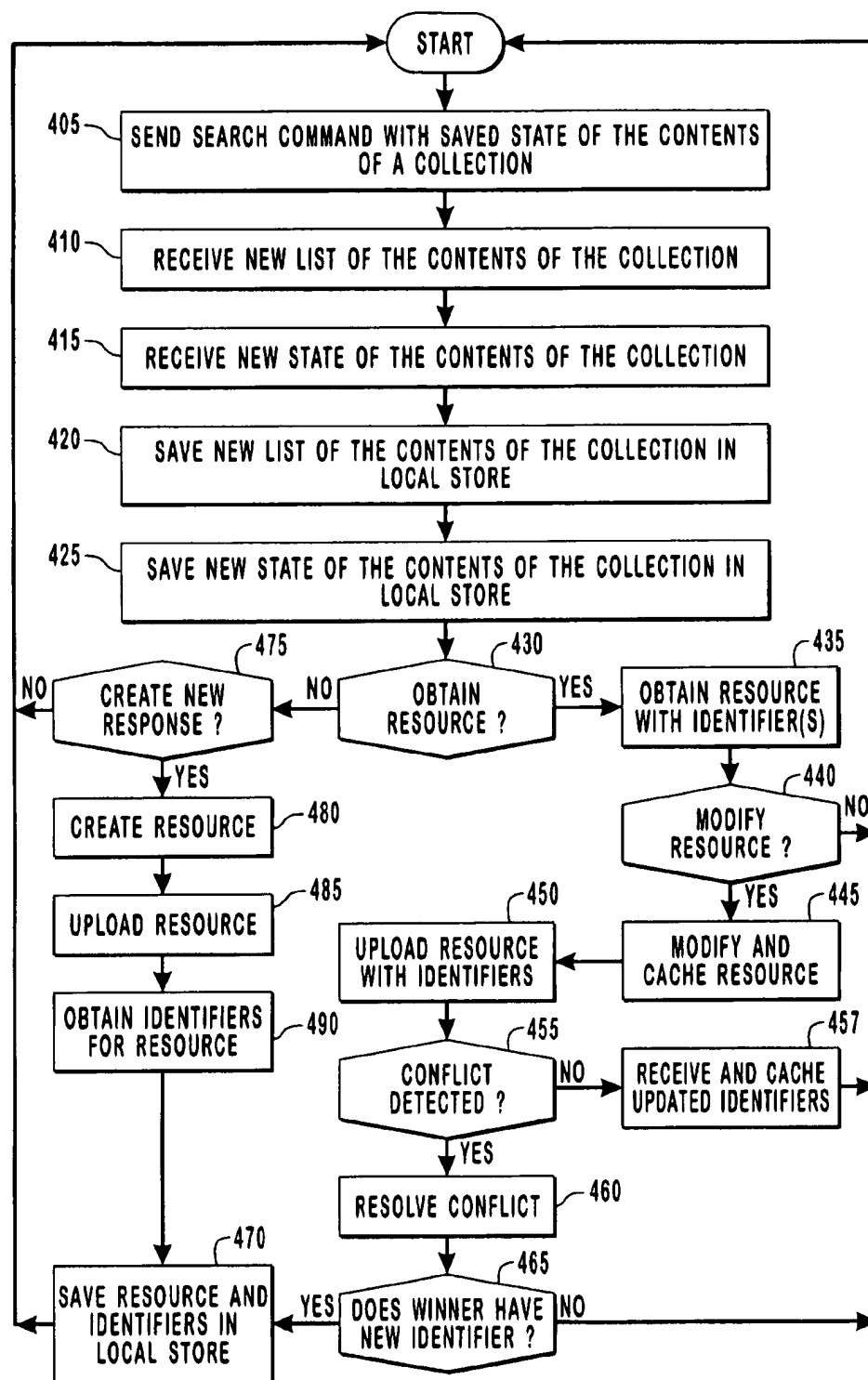


FIG. 4

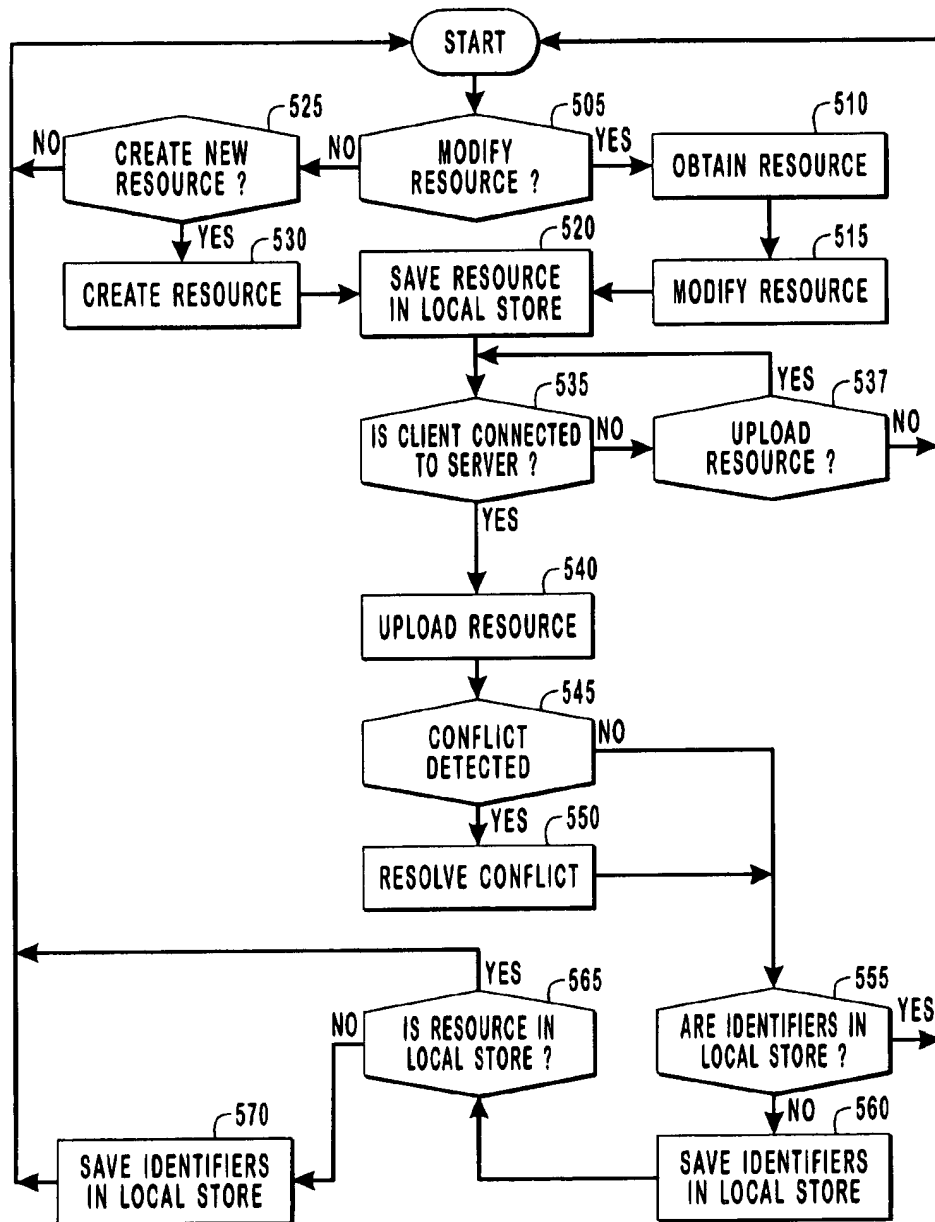


FIG. 5

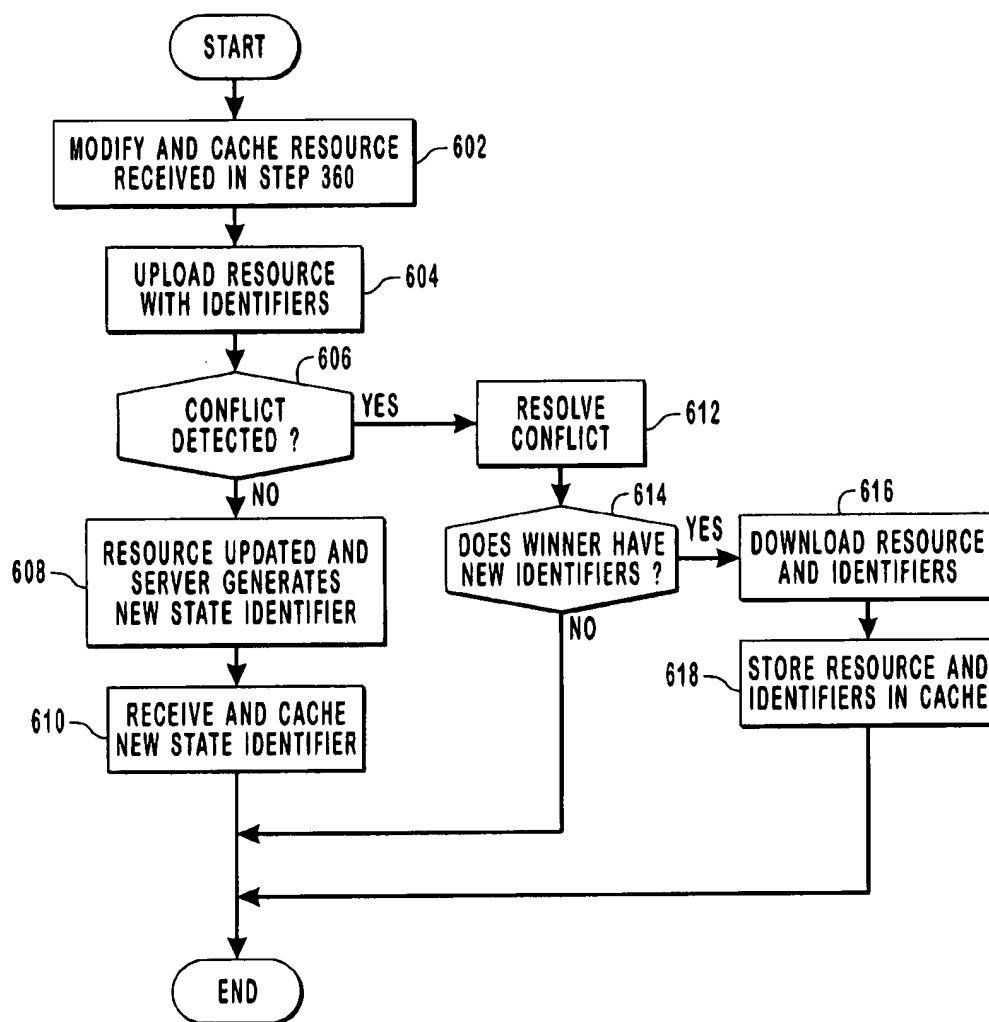


FIG. 6

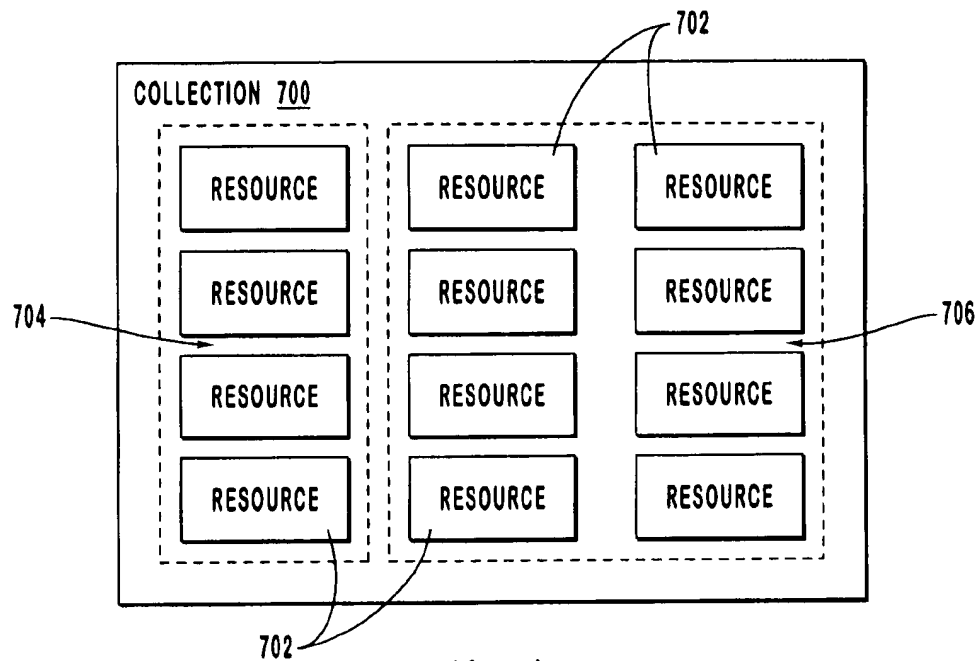


FIG. 7A

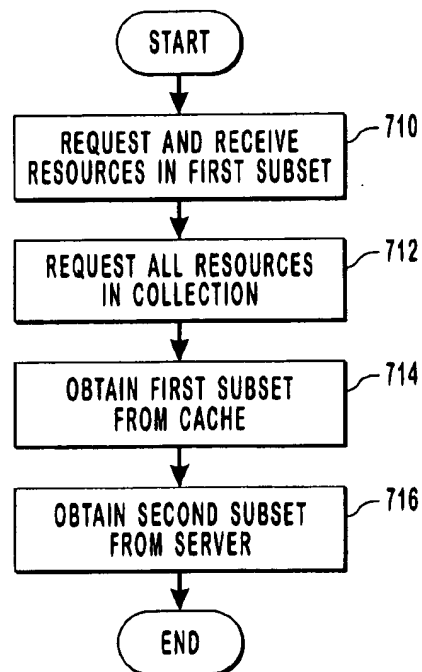


FIG. 7B

METHOD AND SYSTEM FOR SUPPORTING OFF-LINE MODE OF OPERATION AND SYNCHRONIZATION USING RESOURCE STATE INFORMATION

BACKGROUND OF THE INVENTION

1. The Field of the Invention

The present invention relates to the support of on-line and off-line transmission and synchronization of data. More specifically, the present invention relates to systems and methods that eliminate redundant data transmission and allow multiple copies of data to be synchronized so that incremental changes made to one copy of the data can be identified, transferred, and incorporated into the other copy of the data, regardless of whether the incremental changes are made on-line or off-line.

2. The Prior State of the Art

With the advent of the personal computer as the standard information tool, individuals everywhere at anytime need access to information. Never before has there been so much information readily available or such high expectations for how much the individual will be able to accomplish by utilizing information. It is now more common than ever for multiple users to simultaneously, or in tandem, work on shared data such as, by way of example, word processing documents, electronic mail messages, spreadsheets, electronic forms, graphic images, or a host of other data objects. Thus, accessing and sharing current and accurate information among multiple users has become increasingly vital to businesses and individuals.

Traditionally, a user desiring to access shared information located on one or more servers has had to work "on-line." More specifically, the user logs on to a network and communicates directly with a server. All requests from the user are sent directly to the server. Prior to disconnecting the communication, the user updates the data object on the server and must request a local copy or else the user will not have access to the data object once he or she logs off the network. Once "off-line," the user has no way of knowing whether the local copy of the data object contains current and accurate information because the data object on the server could have been modified by a second user while the first user was off-line. Furthermore, if the data object on the server was modified by a second user while the first user was off-line and if the first user makes changes to his/her local copy of the data object, logs on to the network, and saves the local copy on the server under the original name of the data object, changes made to the data object by the second user will be overwritten and lost.

Another limitation is highlighted when two or more users desire access to a data object simultaneously. Under the traditional method, a first user is allowed to obtain and edit the data object. Any subsequent users desiring access to the data object are limited to a "read only" access of the data object. Therefore, while the first user is using the data object, the subsequent users can access but cannot edit the data object. In order to edit, a subsequent user would be required to save the data object under a different name. Once the subsequent user completes the edits, the changes made to the data object by the first user will be overwritten and lost if the changes made by the subsequent user are saved on the server under the original name. Alternatively, if the subsequent user saves the data object on the server under a new name, multiple copies of the data object will exist on the server. One copy will contain the changes made by the first user and

a second copy will contain the changes made by the subsequent user. Where multiple copies exist, it would be difficult to know which copy, if any, contains the most current and accurate information.

Using conventional techniques, clients and servers have engaged in redundant communication of information when a user has been making changes to a shared data object while the client is on-line and subsequently desires to obtain a local copy of the shared data object for use off-line. In particular, the on-line changes made to the shared data object are transmitted from the client to the server during the on-line operation of the client. When the client is about to go off-line, the client issues a request to the server for the most current copy of the shared data object in order to store the copy locally for off-line use. In response, the server transmits the current copy of the data object, which includes the changes that have recently been made by the client. In other words, the foregoing client/server communication involves changes being sent from client to server and subsequently from server to client. Such repetitive transmission of data in a network can introduce potentially significant increases in network traffic, particularly in large organizations with many clients.

It would therefore be desirable to ensure current and accurate information through a model that would synchronize all copies of a data object. It would also be desirable for a synchronization model to be able to identify which copy of a data object is more current and accurate. It would be desirable for none of the changes to be lost or overwritten when multiple copies of a data object are used to update the copy on the server. There is also a need in the art for any such synchronization model to allow multiple users to access and edit the data object simultaneously. Furthermore, it would be desirable if the synchronization mode could eliminate the creation of redundant copies of the data object. It would also be advantageous to eliminate redundant transmissions of data object between clients and servers. Any synchronization model that could exhibit such capabilities would be particularly useful if it could support changes made to local copies of data objects regardless of whether a client was on-line or off-line with the server.

SUMMARY OF THE INVENTION

The foregoing problems in the prior state of the art have been successfully overcome by the present invention, which is directed to a system and method for elimination of redundant data transmission and for incremental change synchronization between multiple copies of data. The systems and methods of the present invention allow multiple copies of data to be synchronized so that incremental changes made to one copy of the data can be identified, transferred, and incorporated into the other copy of the data. The systems and methods of the present invention produce a more efficient use of time and network bandwidth by eliminating the transmission of redundant data. Furthermore, copies of a data object can be reliably and efficiently synchronized regardless of whether the changes made to a copy of the data object are executed while the client is operating on-line or off-line.

Implementation of the present invention may take place in an environment where multiple copies of a data object or resource are present, or where a data object or resource is shared. By way of example, the present invention can be implemented in an environment where a client is connected to a server. The present invention can also be implemented in an environment where multiple clients are connected to

3

the same server. Another example would include multiple clients connected to multiple servers. In each of the environments, the present invention preserves incremental changes to data object regardless of whether the changes are made while a client is on-line or off-line, ensures that all copies of a data object or resource are synchronized without loss of any incremental changes, and prevents the transmission of redundant data between servers and clients.

As part of the present invention, a client, while connected to a server, identifies to the server the current state of data located at the client and issues a request for the server to evaluate the state of the client's data. The server responds to the request by returning an identification of server data that is not included in the client's data and an identification of the client's data that has been changed on the server. The client is then able to download from or upload to the server new or modified data.

As data is uploaded to or downloaded from the server, the data is saved in a local storage location associated with the client and remains available to the user after the client is no longer connected to the server. In other words, as a client interacts with a server, the client stores a copy of the data object or resource and corresponding identification in cache. Therefore, when the client is "off-line" from the server, the client is still able to work on the data object or resource as if it were "on-line" because a copy is located in the client's cache. Furthermore, the presence of the data object or resource copy in cache prevents the need to download a copy from a server each time that the user desires to obtain the data object or resource. Thus, the repetitive transmission of the same data between a client and server is reduced or eliminated according to the invention.

Later, when the client is again "on-line," all copies of the data object or resource are synchronized. The client identifies to the server the current state of the copy in the client's cache. The server determines if the copy in the client's cache is the most current version of the data object or resource, and all copies of the data object are synchronized to the most current version. If there are conflicts between copies of the data object or resource, the conflicts are detected and resolved and all copies are synchronized to reflect the most current version of the data object or resource.

Additional objects and advantages of the invention will be set forth in the description which follows, and in part will be obvious from the description, or may be learned by the practice of the invention. The objects and advantages of the invention may be realized and obtained by means of the instruments and combinations particularly pointed out in the appended claims. These and other objects and features of the present invention will become more fully apparent from the following description and appended claims, or may be learned by the practice of the invention as set forth herein-after.

BRIEF DESCRIPTION OF THE DRAWINGS

In order that the manner in which the above recited and other advantages and objects of the invention are obtained, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments thereof that are illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered to be limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

4

FIG. 1A illustrates an exemplary system that provides a suitable operating environment for the present invention;

FIG. 1B is a schematic illustration of an exemplary operating environment that highlights the use of one or more leaf nodes connected to a cloud of servers for implementation of the present invention;

FIG. 2 is a schematic illustration of data transmission between a client and a server that employs a caching mode of operation;

FIG. 3 is a flowchart illustrating an initial synchronization and download during an on-line mode of operation;

FIG. 4 is a flowchart illustrating a method of resource modification, synchronization, download and/or upload performed after the steps of FIG. 3 during an on-line mode of operation; and

FIG. 5 is a flowchart illustrating a method of off-line resource modification and subsequent resource synchronization that is performed after the steps of FIG. 3.

FIG. 6 is a flowchart depicting a method of modifying the resource downloaded in FIG. 3 and synchronizing the modified resource with a copy stored at the server.

FIG. 7A is a schematic diagram illustrating a collection and resources contained by the collection.

FIG. 7B is a flowchart illustrating a method for efficiently downloading a complete copy of the resources in the collection of FIG. 7A in response by a request by a client.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention extends to systems and methods for the elimination of redundant data transmission in a computer network and for synchronization between multiple copies of data. Multiple copies of data are synchronized so that incremental changes made to one copy of the data can be identified, transferred, and incorporated into all other copies of the data. Eliminating the transmission of redundant data produces a more efficient use of time and network bandwidth. Furthermore, the systems and methods of the present invention support on-line and off-line modes of operation while preserving data transmission efficiency and ensuring reliable synchronization of data.

Embodiments for implementation of the systems and methods of the present invention may comprise a special-purpose or general-purpose computer including computer hardware and/or software components that further include computer-readable media for carrying out or containing computer-executable instructions or data structures. Computer-executable instructions include, for example, instructions and data which cause a special-purpose or general-purpose computer to perform a certain function or group of functions. Computer-readable media can include any available media that can be accessed by a general purpose or special purpose computer. By way of example, and not limitation, such computer-readable media can comprise RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other medium that can be used to carry or store desired program code means in the form of computer-executable instructions or data structures, or that can be used to store information or data, and that can be accessed by a special-purpose or general-purpose computer.

When information is transferred or provided over a means of communication, such as a network (either hardwired, wireless, or a combination of hardwired or wireless), to a computer, the computer properly views the connection as a

5

computer-readable medium. Thus, any such connection can be properly termed a computer-readable medium.

For purposes of illustration, this description of the invention refers to diagrams and flowcharts depicting the structure or processing of embodiments for implementing the systems and methods of the present invention. Using the diagrams and flowcharts in this manner should not be construed as limiting the scope of the present invention. The description of the invention presented herein makes reference to terminology, methods and concepts from RFC 2518. Therefore, for purposes of the description, RFC 2518 is incorporated herein by reference. Examples of terms used herein and defined in RFC 2518 include "resource" and "collection."

FIGS. 1A and 1B provide suitable computing environments in which the invention may be implemented. Although not required, the invention will be described in the general context of computer-executable instructions, such as program modules, being executed by computers in network environments. Generally, program modules include routines, programs, objects, components, data structures, and so forth that perform particular tasks or implement particular abstract data types. Computer-executable instructions, associated data structures, and program modules represent examples of the program code means for executing steps of the methods disclosed herein. The particular sequence of such executable instructions or associated data structures represent examples of corresponding acts for implementing the functions described in such steps.

Those skilled in the art will appreciate that the invention may be practiced in network computing environments with many types of computer system configurations, including personal computers, hand-held devices, multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, and the like. The invention is described herein in reference to distributed computing environments where tasks are performed by local and remote processing devices that are linked (either by hardwired links, wireless links, or by a combination of hardwired or wireless links) through a communications or computer network. In the distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to FIG. 1A, an exemplary system for implementing the invention includes a general purpose computing device in the form of a conventional computer 20, including a processing unit 21, a system memory 22, and a system bus 23 that couples various system components including the system memory 22 to the processing unit 21. The system bus 23 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system (BIOS) 26, containing the basic routines that help transfer information between elements within the computer 20, such as during start-up, may be stored in ROM 24.

The computer 20 may also include a magnetic hard disk drive 27 for reading from and writing to a magnetic hard disk 39, a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading from or writing to removable optical disk 31 such as a CD-ROM or other optical media. The magnetic hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to the system bus 23 by a hard disk

6

drive interface 32, a magnetic disk drive-interface 33, and an optical drive interface 34, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer-executable instructions, data structures, program modules and other data for the computer 20. Although the exemplary environment described herein employs a magnetic hard disk 39, a removable magnetic disk 29 and a removable optical disk 31, other types of computer readable media for storing data can be used, including magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, RAMs, ROMs, and the like.

Program code means comprising one or more program modules may be stored on the hard disk 39, magnetic disk 29, optical disk 31, ROM 24 or RAM 25, including an operating system 35, one or more application programs 36, other program modules 37, and program data 38. A user may enter commands and information into the computer 20 through keyboard 40, pointing device 42, or other input devices (not shown), such as a microphone, joy stick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 21 through a serial port interface 46 coupled to system bus 23. Alternatively, the input devices may be connected by other interfaces, such as a parallel port, a game port or a universal serial bus (USB). A monitor 47 or another display device is also connected to system bus 23 via an interface, such as video adapter 48. In addition to the monitor, personal computers typically include other peripheral output devices (not shown), such as speakers and printers.

The computer 20 may operate in a networked environment using logical connections to one or more remote computers, such as remote computers 49a and 49b. Remote computers 49a and 49b may each be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 20, although only memory storage devices 50a and 50b and their associated application programs 36a and 36b have been illustrated in FIG. 1A. The logical connections depicted in FIG. 1A include a local area network (LAN) 51 and a wide area network (WAN) 52 that are presented here by way of example and not limitation. Such networking environments are commonplace in office-wide or enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment, the computer 20 is connected to the local network 51 through a network interface or adapter 53. When used in a WAN networking environment, the computer 20 may include a modem 54, a wireless link, or other means for establishing communications over the wide area network 52, such as the Internet. The modem 54, which may be internal or external, is connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules depicted relative to the computer 20, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing communications over wide area network 52 may be used.

Now referring to FIG. 1B, an exemplary network environment shown generally at 102 is illustrated. Network 102 includes a cloud of servers 100 that provides network and computing services. The term "cloud of servers" is to be understood as referring to a group of one or more servers, any one of which may be used to provide network and computing services to clients. For purposes of illustration, cloud 100 includes three servers, namely, servers 110, 120, and 130. The number of servers in any particular network in

which the invention is practiced is not critical with respect to the manner in which the invention operates. Depending on the size of network 102, the capabilities of the servers, and the network services required by the clients, as few as one server or as many servers as needed may be included in cloud 100. Moreover, the individual servers 110, 120 and 130 in cloud 100 may be in a single general location in an enterprise, or may be located remotely with respect to one another.

Network 102 also includes one or more clients connected to the cloud of servers 100 by way of a data communication infrastructure 180, which may be wired or wireless. For illustration purposes, clients 140, 150, 160, and 170, which may be any special-purpose of general-purpose computers, are depicted in FIG. 1B. As used herein, the term "leaf nodes" shall refer to clients associated with a network in which the invention is employed. In practice, network 102 can have as few as one client or can have as many clients as is required by the enterprise or organization served by the network. While network 102 might have only one client, the full benefits of the invention are often more pronounced when a plurality of clients are associated with the network, since the invention enables access and synchronization of multiple copies of a particular resource that may be shared among any number of clients. Indeed, one of the advantageous features of the invention is that it can reliably and efficiently manage the synchronization of shared resources, even as the number of clients associated with network 100 grows large.

A client can comprise an internal or external computer readable media for providing local, nonvolatile storage of information, as will be further explained below. Servers 110, 120 and 130, are kept in consistent replica of data among themselves so that changes can be entered into any one of the servers. As will be explained below, implementation of the present invention allows for the modifying of a resource by a client regardless of whether that client is logged on or logged off from communication infrastructure 180, ensures synchronization of all copies of a resource whether at a server or at a client, and optimizes network bandwidth and time by preventing redundant transmission of data across infrastructure 180.

FIG. 2 illustrates an embodiment of the present invention wherein data is transmitted between a server and a client that employs a caching mode of operation. As mentioned above, a client, such as client 140, can comprise an internal or external computer readable media, which may be a nonvolatile storage device providing local storage of information. FIG. 2 illustrates the computer readable media as local store 200.

FIG. 2 illustrates an embodiment of the invention associated with a caching mode of operation in which data is transmitted from client 140 to any one of the servers within the cloud of servers 100. The data can be sent either by way of direct transmission, or by first passing the data through local store 200, as depicted by the combination of transmissions 210 and 220. In a further embodiment, when client 140 is on-line with any one of the servers within the cloud of servers 100, all data transmission is sent directly to the server. When client 140 is off-line with servers 100, all data operations, such as read requests, write requests, delete requests and other communication generated for accessing or manipulating a resource is communicated between client 140 and local store 200.

As used herein, the term "on-line" refers to a state in which a client communicates with a server using the com-

munication infrastructure. For example, a client that is sending network requests to a server or receiving network or computing services from the server is on-line. In contrast, the term "off-line" refers to a client is operating in a state in which it is not immediately capable of actively communicating with a server. For instance, a client that has physically disconnected from a wired communication infrastructure or is not communicating over a wireless communication infrastructure is off-line. Moreover, a client that may be physically connected to a communication infrastructure, but is operating only locally without communicating over the infrastructure can also be considered to be off-line.

In contrast to the communication from client to server illustrated in FIG. 2, all data transmission originating from a server is first passed through local store 200 before reaching client 140 in this embodiment. In this manner, local store 200 can cache all data transmitted from the cloud of servers 100 to client 140. As such, when client 140 is off-line, it can obtain the data cached in local store 200 in a similar way as if client 140 were on-line with any one of the servers within the cloud of servers 100, such as server 110, and obtained the data directly from server 110. Therefore, when client 140 requests data that has been cached in local store 200, it can obtain the data regardless as to whether client 140 is on-line or off-line. Indeed, from the standpoint of client 140, retrieval of a resource from local store 200 during an off-line mode of operation can appear as if the client remained on-line and were accessing the resource from servers 100.

Another embodiment of the invention is associated with a caching mode of operation. In this embodiment the data can only be sent to one of the servers, such as server 110, by first sending the data to local store 200, as depicted by the combination of transmissions 210 and 220. Therefore, local store 200 is involved in all communication with servers 100 while client 140 is on-line and all read requests, write requests, or any other communication generated for accessing or manipulating a resource while the client is off-line.

Similarly, all data transmission originating from a server is first passed through local store 200 before reaching client 140. In this manner local store 200 can cache all data transmitted from the cloud of servers 100 to client 140. As such, when client 140 is off-line, it can obtain the data cached in local store 200 in a similar way as if client 140 were on-line with any one of the servers within the cloud of servers 100. Therefore, when client 140 requests data that has been cached in local store 200, it can obtain the data regardless as to whether client 140 is on-line or off-line.

As introduced above, the systems and methods of the present invention provide for the synchronization between multiple copies of data. Multiple copies of data are synchronized so that incremental changes made to one copy of the data can be identified, transferred, and incorporated into all other copies of the data.

FIG. 3 illustrates one embodiment of a method for performing initial synchronization between a client and server and download during an on-line mode of operation. The method of FIG. 3 can be used to coordinate the use and modification of any type of resource. However, in order to establish a context in which the following description can be easily understood and not by limitation, the resources could be a set of e-mail messages stored at the server, while the collection could be an inbox that stores the e-mail messages or the resources could be a set of word processing documents, while the collection could be a file system folder that stores the word processing documents. In the following

description associated with FIGS. 3-5, specific reference will be made to word processing documents to illustrate this embodiment of the invention. Again, however, the invention can be practiced with any desired type of resource.

In step 310 a client requests a list of the contents of a collection located at a server. The server then responds by transmitting a list of the resources contained in the collection and information that specifies the state of the resources contained in the collection, as respectively illustrated by steps 320 and 330. In step 320 the client receives the list of the contents of the collection. In one embodiment, the list includes identifiers representing each of the resources of the collection. Although not necessary, the identifiers can be compressed prior to transmission to optimize network bandwidth and time. The identifiers can include, for example, a Uniform Resource Identifier (URI) or other information that uniquely identifies the resources.

In step 330 the client receives the information specifying the state of the contents of the collection. The "state" of the contents of a collection stored at a server refers to the identity of the current version of a resource stored at a server. The state of any particular copy of a resource refers to the version of the resource when it was stored at the server. As a particular resource stored at one or more servers undergoes a series of successive updates, the resource is considered to have passed through a corresponding series of states, each of which represents a single update version of the resource. In one embodiment, information specifying the state of the contents of the resources includes an identifiers, which can be termed resource state identifiers. Although not necessary, these identifiers can also be compressed to optimize network bandwidth and time.

In summary, steps 310, 320, and 330 result in the client being given at least two pieces or sets of information associated with the collection specified in the request of step 310. First, the client is given information representing the identity of the various resources that are contained in a collection stored at the server in step 320. Second, the client is given information that essentially represents or is associated with the current update version of the various resources that are contained in the collection in step 330. These two pieces or sets of information can be subsequently used by the methods and systems of the invention to manage synchronization of copies of the resources contained in the collection and to eliminate repetitive transmission of resources or portions of resources between the client and the server as will be further described herein below.

In order to obtain a more detailed description of the processes and mechanisms whereby steps 310, 320, and 330 can be performed, reference is made to U.S. patent applications Ser. No. 09/412,739, entitled "Method, Computer Readable Medium, and System For Monitoring The State of A Collection of Resources", filed on Oct. 4, 1999, which assigned to the same assignee as the present application, and which is incorporated herein by reference; and Ser. No. 09/412,071, entitled "Method, Data Structure, and Computer Program Product For Identifying A Network Resource", filed on Oct. 4, 1999, which assigned to the same assignee as the present application, and which is incorporated herein by reference.

In steps 340, 350 and 370 data is cached in a local storage associated with the client. More specifically, in step 340, the list of the contents of the collection received in step 320 is cached in a local storage. In step 350 the state of the contents of the collection received in step 330 is cached in local storage. The client can then obtain one or more resources

from the server. In step 360 the client requests and the server downloads a resource and corresponding identifiers in the collection. If, for example, the resources in the collection are word processing documents, the client in 360 can request a selected document. The corresponding identifiers referred to in step 360 can include information that represents the current update version of the downloaded resource. In step 370 the downloaded resource and corresponding identifiers are cached in local storage. For purposes of this description and the accompanying claims, the term "download" refers to data transmission from server to client, whereas "upload" refers to data transmission from client to server.

It should be noted that steps 360 and 370 are optional, depending on whether the user of the client wants to obtain a particular resource at this time. If no resource is downloaded at this time, the client still has obtained information in steps 320 and 330 that identifies the resources contained in the collection and represents the current update version of the resources as they are stored at the server.

FIGS. 4-6 represent various methods and operations that can be performed in the network after the initial synchronization of FIG. 3. FIG. 4 represents a method of performing on-line modification, downloading, uploading, and synchronization of resources. FIG. 5 depicts a method whereby a client modifies a resource off-line, which is later synchronized on-line. Finally, FIG. 6 represents a method of modifying and synchronizing the resource downloaded in step 360 of FIG. 3.

FIG. 4 provides a flowchart illustrating a subsequent resource modification, synchronization, download and/or upload during an on-line mode of operation. The mode of operation in FIG. 4 is "on-line" in the sense that any modification of resources is conducted while the client is on-line. The method illustrated in FIG. 4 is conducted after an initial synchronization has been conducted as shown, for example, in FIG. 3 and assumes that the client operates in the caching mode.

In step 405 of FIG. 4, the client again requests a list of the contents of a collection located at the server. The server then responds by transmitting the list and state of the contents of the collection, as respectively illustrated by steps 410 and 415. In step 410 the client receives a new list of the contents of the collection. Depending on the activity associated with the collection that has been conducted at the server since the time of the previous request (i.e., step 310 of FIG. 3), the list obtained in step 410 can be identical or different from the list obtained in step 320 of FIG. 3. For instance, if a word processing document has been deleted or created at the server (perhaps by a different client) since the time of step 310, the list obtained in step 410 will include information identifying the new word processing document. In contrast, if no changes to the collection have been made, the list received in step 410 will be identical to the list obtained in 320.

In step 415 the client receives a new state of the contents of the collection. In a similar manner, the new state can be identical to the state previously received in step 330 of FIG. 3 or can be an updated version. For example, if an existing word processing document has been modified and stored at the server (perhaps by a different client) since the time of the previous request (i.e., step 310), the state of the modified word processing document will be different, reflecting the updated version of the document. Similarly, if a word processing document has been created or deleted at the server since the time of step 310, the state of the contents of the collection will also be different. In contrast, if no updated

11

versions have been stored and no resources have been created or deleted, the state received in step 415 will be identical to the list received in step 330.

Assuming that the new list and the new state are different from the previous list and state, steps 420 and 425 cache the new list and new state, respectively, in local storage. Decision block 430 then inquires as to whether the client desires to obtain a resource from the server. If the client does, the method advances to step 435. Otherwise, the method proceeds to decision block 475.

In step 435 the client requests a resource and the identifiers corresponding to the resource. In this step and others that follow, unless otherwise indicated, the "identifiers" refer to at least information that uniquely identifies the resource and information that specifies the update version of the resource. If the resource has never been cached in local storage, the resource and corresponding identifiers are downloaded from the server in step 435. In this manner, repetitive downloading of the same resource to a particular client is eliminated.

Alternatively, if the resource has been previously cached in the local store associated with the requesting client, step 435 involves a comparison that takes place between the state of the resource in the local store with the state of the resource on the server. This comparison is conducted in order to determine whether the copy of the resource stored in the local store is the most recent version or whether a more recent version exists at the server. In other words, the comparison addresses the possibility that another client has modified and updated the requested resource since the last time that the requesting client has obtained the copy of the resource. It should be noted that the comparison requires transmission of an identifier representing the state of the resource, without requiring transmission of the entire resource between client and server. In this manner, the comparison reduces the network traffic that might otherwise be required and avoids transmitting the same version of the resource more than once.

A more detailed explanation of how the comparison can be performed is included in U.S. patent application Ser. No. 09/412,739 entitled "Method, Computer Readable Medium, and System For Monitoring The State of A Collection of Resources", which was previously incorporated herein by reference; and in U.S. patent application Ser. No. 09/412,738, entitled "Systems and Methods for Detecting and Resolving Resource Conflicts" which was filed on Oct. 4, 1999, which was assigned to the same assignee as the present application, and which is incorporated herein by reference.

If the resource at the local store is found to have an identical or more recent state compared to the state of the resource located at the server, then the client obtains the resource in step 435 by accessing the locally stored resource rather than the copy stored at the server, thereby optimizing network bandwidth and time. Otherwise, the resource and corresponding identifiers are downloaded in step 435 from the server and cached in the local store, thereby replacing the stale resource and identifiers that have been stored locally.

Decision block 440 inquires as to whether the client is to modify the resource. For instance, the client is to modify the resource if the user wants to edit the contents of the resource or delete the resource. If the client does not desire to modify the resource, processing returns back to step 405. Otherwise, the client modifies the resource in step 445 and it is cached in local storage.

After the modifications have been performed, the client in step 450 can update the copy of the resource at the server by

12

uploading the resource with its corresponding identifiers. Upon receiving the uploaded resource and corresponding identifiers, the server in decision block 455 compares the state of the uploaded resource with the copy of the resource located at the server to determine whether there is a conflict. In other words, the comparison determines whether the copy of the resource stored at the server has been modified during the period of time that has elapsed since the uploading client last accessed the resource from the cloud of servers. To illustrate further, a conflict may arise when two clients access the same copy of a word processing document stored at a server and both attempt to make and save modifications to the document.

If there is no conflict, the server updates the version of the resource and the identifiers stored at the server and, in step 457, the updated identifiers are received by the client and cached in local storage. Processing then returns to step 405. If a conflict is detected then the conflict is resolved in step 460. A server or a client can perform conflict resolution. Alternatively, a user can be prompted to resolve the conflict. A more detailed description of conflict resolution that can be used with the invention is included in U.S. patent application Ser. No. 09/412,738, entitled "Systems and Methods for Detecting and Resolving Resource Conflicts." Once the conflict has been resolved, decision block 465 inquires as to whether the winning, or more current, resource has corresponding identifiers that are different from the corresponding identifiers located in cache. If the identifiers are identical then processing returns to step 405. Otherwise, the winning resource and corresponding identifier are cached in local storage in step 470.

Returning to decision block 430, if it is determined that the client does not desire to obtain a resource then decision block 475 determines whether the client desires to create a new resource. If the client does not desire to create a new resource then processing returns to step 405. Otherwise, the client creates a new resource in step 480 and the new resource is uploaded to the client in step 485. In step 490 the server assigns identifiers to the new resource and downloads the identifiers to the client. It should be noted that, in this embodiment, the identifiers assigned by the server to the new resource is downloaded to the client in step 490, but the resource is not downloaded, since it already exists at the client. The new resource and corresponding identifiers are cached in local storage in step 470 and processing returns to step 405.

FIG. 5 provides a flowchart illustrating resource modification, synchronization, download and/or upload, some of which can occur during an off-line mode of operation. The mode of operation in FIG. 5 is "off-line" in the sense that modification of resources can be conducted while the client is off-line. For instance, a network user can download a resource, disconnect the client from the network, make modifications to the resource while off-line, and later synchronize while on-line. The method illustrated in FIG. 5 is conducted after an initial synchronization is performed on-line as shown, for example, in FIG. 3. FIG. 5 further assumes that the client operates in the caching mode.

For purposes of illustration, it is presumed that the client begins off-line at decision block 505, which 505 inquires as to whether the client desires to modify a resource. If the client does not desire to modify the resource, then processing proceeds to decision block 525. Otherwise, the client requests and obtains a resource in step 510. Because a presumption is being made that the client is off-line, the client obtains the copy of the requested resource stored in the local store associated with the client in step 510, if such a

13

copy is stored locally. If the resource is not yet cached in the local store, an error is displayed and processing returns to decision block 505. The client modifies the resource in step 515 and it is cached in step 520.

Returning briefly to step 510, if the client had instead been on-line and the requested resource had been stored locally, a comparison would have been made regarding the state of the local copy of the resource and the state of the copy of the resource stored at the server. If the state of the resource stored at the server demonstrated that the server's copy of the resource was more current than the cached copy of the resource, the server's copy and corresponding identifiers would have been downloaded to the client and cached.

Returning now to the off-line presumption and to decision block 505, if the client does not desire to modify the resource, decision block 525 inquires as to whether the client desires to create a new resource. If a new resource is not to be created then processing returns to decision block 505. Otherwise, the new resource is created in step 530 and cached in step 520.

After the resource is cached in step 520, decision block 535 inquires as to whether the client is on-line. If it is not on-line, decision block 537 asks whether the client desires to upload the resource to the server. Processing returns to decision block 505 if the resource is not to be uploaded. Otherwise, processing remains in a loop between decision blocks 535 and 537 until the client is on-line with the server.

Once the client is on-line, the client uploads the resource and the corresponding identifiers to the server in step 540. Upon receiving the uploaded resource and corresponding identifiers the server in decision block 545 compares the state of the uploaded resource with the copy of the resource located at the server, if any. If there is no conflict then the server updates the identifiers and decision block 575 asks whether the corresponding identifiers have been cached.

If a conflict is detected in decision block 545 then the conflict is resolved in step 550. A server or a client can perform conflict resolution. Alternatively, a user can be prompted to resolve the conflict. A more detailed description of conflict resolution that can be used with the invention is included in U.S. patent application Ser. No. 09/412,738, entitled "Systems and Methods for Detecting and Resolving Resource Conflicts."

Decision block 555 inquires as to whether the identifiers corresponding to the winning, or more current, resource are located in cache. If the identifiers are in cache then processing returns to decision block 505. Otherwise, the identifiers are cached in step 560. Decision block 565 inquires as to whether the winning, or more recent, resource is in cache. If it is then processing returns to decision block 505. Otherwise the resource is cached and processing returns to decision block 505.

FIG. 6 illustrates selected steps of a method for modifying and synchronizing the resource that was received by the client in step 360 of FIG. 3. The method illustrated in FIG. 6 is performed after an initial synchronization has been conducted as shown, for example, in FIG. 3 and assumes that the client operates in the caching mode. Step 602 can be executed while the client is on-line or off-line.

In step 602, the client, either in the on-line or off-line mode, modifies the resource received in step 3 of FIG. 3 and stores the changes to the resource in cache in the local store. Later, when the client is on-line and is to initiate synchronization of the resource, it uploads the resource and the corresponding identifiers, including the information (obtained in step 360) that specifies the update version of the copy of the resource stored at the server.

14

In decision block 606, the server then performs conflict detection as disclosed previously in reference to step 455 of FIG. 4. If no conflict is detected or, in other words, if the copy of the resource to be overwritten at the server has not changed since the time it was downloaded to the client in step 360 of FIG. 3, the server updates the resource in step 608 and generates a new state identifier, which specifies that a new updated version of the resource has been stored at the server. In step 610, the client receives and caches the new state identifier. It is noted that the server does not need to transmit the updated copy of the resource in this embodiment, since the client has already cached this version of the resource in step 602.

Returning to decision block 606, if a conflict is detected, the method advances to step 612, in which the conflict is resolved. A server or a client can perform conflict resolution. Alternatively, a user can be prompted to resolve the conflict. A more detailed description of conflict resolution that can be used with the invention is included in U.S. patent application Ser. No. 09/412,738, entitled "Systems and Methods for Detecting and Resolving Resource Conflicts." Once the conflict has been resolved, decision block 614 inquires as to whether the winning, or more current, resource has new identifiers that are different from the corresponding identifiers located in cache. If the identifiers corresponding to the winning resource are not new, the method illustrated in FIG. 6 ends. Otherwise, the winning resource and corresponding identifiers are downloaded to the client in step 616 and cached in local storage in step 618.

FIGS. 7A and 7B illustrate one example whereby the invention efficiently uses time and network bandwidth. FIG. 7A depicts a collection 700 that has a plurality of resources 702 contained therein that can be stored at the one or more servers included in the cloud of servers depicted in FIG. 1B. For purposes that will become clear below in reference to FIG. 7B, resources 702 are arbitrarily divided into two mutually exclusive subsets, namely, subset 704 and subset 706. As noted previously, the invention can be practiced with any desired collections and resources. However, in order to establish an clearly understandable context in which FIG. 7B can be described, it will be assumed that resources 702 are e-mail messages, while collection 700 is an inbox that contains the e-mail messages.

FIG. 7B is a flowchart illustrating an example in which a client has been accessing a first set of resources in a collection while on-line and, in preparation for going off-line, requests a copy of all the resources in a collection. Rather than download all of the resources in the collection, the server downloads only those that have not yet been accessed by the client, since the previously accessed resources are cached in the local store associated with the client. The server uses the techniques disclosed above in reference to FIGS. 3-6 to determine which resources have been previously accessed by the client and which are to be downloaded, and these techniques are further described below in reference to the specific steps of FIG. 7B.

In step 710, the client requests only selected resources in collection 700. One specific example of selecting only some resources in a collection might happen when the user of the client is interested in accessing only specific e-mail messages stored in an inbox. For instance, in step 710, the client might request the first subset of resources 704 illustrated in FIG. 7A. While not explicitly illustrated in FIG. 7A, the server transmits the identifiers disclosed above in reference to steps 310, 320, and 330 of FIG. 3 during step 710 for all of the resources 702 of collection 700. Moreover, the downloaded first set of 323 resources 704 are cached in the

15

local store associated with the client, meaning that these same resources will not need to be sent a second time to the client.

In step 712, the client requests all of resources 702 in collection 700. Such a request might occur, for example, when the user of the client is preparing to go off-line and wishes to have access to all the e-mails in the inbox. To execute the request, the client and server perform the operations disclosed above in reference to step 435. In particular, the client obtains the first subset of resources 704 from the cache and the second subset of resources from the server. Moreover, if any of the resources of the first subset have changed at the server since the time at which step 710 was executed, the server downloads those changed resources to the client as further described above in reference to step 435. In any event, no single version of a resource is sent more than once from the server to the client. In view of this example, it can be clearly seen that the systems and methods of the invention can significantly reduce the time and network traffic that might otherwise be associated with uploading, downloading, and synchronization of resources on a network.

Therefore, as described herein, the systems and methods of the invention eliminate transmission of redundant data by assigning and comparing identifiers for resources to ensure that a resource is transmitted only once to a specific location, thereby producing a more efficient use of time and network bandwidth. The invention also synchronizes multiple copies of a resource by assigning identifiers, comparing identifiers, determining which is the most current resource, and updating the stale resources. Furthermore, no incremental change is lost because resources that are the most current are maintained and because conflicts between resources are detected and resolved. Furthermore, the systems and methods of the present invention support modification of resources while clients are in on-line and off-line modes of operation. Moreover, when a client requests data that has been cached in a local store, it can obtain the data regardless as to whether the client is "on-line" or "off-line."

The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes that come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed and desired to be secured by United States Letters Patent is:

1. In a networked system including a server and a client, wherein the server stores a resource, a method of enabling a client to access the resource, comprising the steps of:

transmitting, from the server to the client, resource state information that represents the state of the resource stored at the server at a selected moment;

transmitting, from the server to the client, a copy of the resource as it exists at the server at the selected moment;

storing the copy of the resource in a local store associated with the client;

modifying the copy of the resource stored in the local store;

transmitting the modified copy and the resource state information from the client to the server;

based on the resource state information transmitted to the server, determining whether the resource stored at the server has changed since the selected moment; and

16

if it is determined that the resource stored at the server has not changed, replacing the resource stored at the server with the modified copy.

2. A method as recited in claim 1, wherein the local store is a nonvolatile storage device that is used as a cache.

3. A method as recited in claim 1, wherein the step of modifying is conducted by the client while the client is off-line with respect to the server.

4. A method as recited in claim 1, further comprising, after the step of replacing, the step of transmitting new resource state information to the client.

5. A method as recited in claim 1, wherein the step of transmitting the modified copy comprises the step of storing the modified copy at the local store without transmitting the modified copy from the server to the client.

6. A method as recited in claim 1, wherein the step of determining whether the resource stored at the server has changed since the selected moment comprises the step of comparing, at the server, the resource state information transmitted to the server with a current version of the resource state information, wherein the current version of the resource state information represents the state of a version of the resource stored at the client at the time of the step of comparing.

7. A method as recited in claim 6, wherein the step of comparing results in a determination that the resource state information transmitted to the server and the current version of the resource state information are the same, thereby determining that the resource stored at the server has not changed.

8. A method as recited in claim 6, wherein the step of comparing results in a determination that the resource state information transmitted to the server and the current version of the resource state information are different, thereby determining that the resource stored at the server has changed, the method further comprising the step of performing a conflict resolution operation in response to the resource stored at the server having changed.

9. A method as recited in claim 1, further comprising the step of storing, in the local store, the resource state information.

10. A method as recited in claim 1, wherein step of transmitting the copy of the resource from the server to the client is performed only once.

11. In a client associated with a networked system that includes a server that stores a resource, a method for accessing the resource and then interacting off-line with the resource in a manner such that it appears, from the standpoint of the client, that the client is on-line, comprising the steps of:

while the client is on-line with the server, receiving from the server a copy of the resource and resource state information representing the state of the resource stored at the server at a selected moment;

storing the copy of the resource in a local store associated with the client;

placing the client in an off-line condition with respect to the server; and

performing a data operation on the copy of the resource while the client is off-line by accessing the copy of the resource in the local store, the data operation resulting in a modified copy of the resource; and

synchronizing the resource stored at the server with the modified copy of the resource while the client is subsequently on-line after the step of performing the data operation, the synchronization being performed at

17

least in part by transmitting to the server the copy or the resource stored at the client, and the resource state information.

12. A method as recited in claim 11, further comprising the steps of:

determining whether the resource stored at the server has changed since the selected moment based on the resource state information transmitted to the server; and if it is determined that the resource stored at the server has not changed, replacing the resource stored at the server with the modified copy.

13. A method as recited in claim 11, wherein the data operation includes a read operation.

14. A method as recited in claim 11, wherein the data operation includes a write operation.

15. A method as recited in claim 11, wherein the data operation includes a delete operation.

16. A computer program product for implementing, in a server included in a network that also includes at least a first client and a second client, a method for synchronizing multiple copies of a resource that are stored at various locations in the network, the computer program product comprising:

a computer-readable medium carrying computer-executable instructions for implementing the method, the computer-executable instructions comprising:

program code means for generating resource state information that represents the current state of a resource stored at the server;

program code means for transmitting, from the server to the first client, a copy of the resource state information as it exists at the server at the selected moment and a copy of the resource as it exists at the selected moment;

program code means for receiving a request to update the resource stored at the server with a modified copy of the resource generated by the first client, the request including the copy of the resource state information;

program code means for determining whether a conflict exists in response to the request by comparing the copy of the resource state information received from the first client with the resource state that represents the current state of the resource stored at the server; and

18

program code means for replacing the resource stored at the server with the modified copy if it is determined that no conflict exists.

17. A computer-readable medium as recited in claim 16, wherein the computer-executable instructions further comprise program code means for performing conflict resolution if it is determined that a conflict exists.

18. A computer-readable medium as recited in claim 16, wherein the request to update the resource is encoded in an XML language.

19. A computer program product for implementing, in a server included in a network that also includes at least a first client and a second client, a method for synchronizing multiple copies of a resource that are stored at various locations in the network, the computer program product comprising:

a computer-readable medium carrying computer-executable instructions for implementing the method, the computer-executable instructions comprising:

program code means for generating resource state information that represents the current state of a resource stored at the server;

program code means for transmitting, from the server to the first client, a copy of the resource state information as it exists at the server at the selected moment and a copy of the resource as it exists at the selected moment;

program code means for receiving a request to update the resource stored at the server with a modified copy of the resource generated by the first client, the request including the copy of the resource state information;

program code means for determining whether a conflict exists in response to the request by comparing the copy of the resource state information received from the first client with the resource state that represents the current state of the resource stored at the server; and

program code means for performing conflict resolution if it is determined that a conflict exists.

20. A computer-readable medium as recited in claim 19, wherein the request to update the resource is encoded in an XML language.

* * * * *